Archive Retrieval Server Level 1 design

From Intranet

This is the second Archive Retrieval Server API design draft. It follows the initial design draft (ADIC Level 0 design). This design incorporates user input regarding the Stornext (formerly called "ADIC") system capabilities desired to serve their needs.

This design will begin by identifying categories of users, then list use cases and then from that derive a list of requirements. This should provide a good outline that will improve with user input.

Contents

User Categories

Identify the players the Retrieval Server API software will serve

End Users
: Most users will come in over the Web. These users will be seeking data for scientific uses, but their background will vary widely both technically and scientifically. The interaction they have will be with the interfaces designed by the data managers, who will provide domain specific background information and search capabilities. A shopping cart type of data ordering system is expected to meet most needs. A handful of internal users will also be served, and will require higher privileges.

  External Users
    standard interactions over the web
    large orders over a set threshold will invoke data manager intervention
  Internal Users (Data managers)
    heightened access
    ability to create larger datasets
    additional data delivery options, including destination location for files

Data Managers
: Internal administrators at NGDC will need a set of capabilities to serve End Users. They will use the Archive Retrieval Server API or direct a programmer to create an interface for Web End Users. Data managers will want to see the status of their datasets, orders requested and delivered, and summary statistics. They can also control orders for external users, such as altering the priority.

Administrative monitors
: Administrative monitors can be system administration personnel that want to track load on the system, as well as NGDC management that will want to know the types and amount of data being delivered as well as summary statistics about the users and system load.

Use Cases

Enumerate some scenarios of usage that the Archive Retrieval Server API should serve

1. External Web User Shopping Cart

A web user is visiting a domain specific page (probably an NGDC page) and would like to access data that are hosted on the Storenext system. A common and familiar shopping cart set of pages will allow the user to select and then download data files.

File directory page

The web user follows a link that shows a directory listing, the starting directory is appropriate to the domain specific page they started with. This listing may be similar to an operating system directory listing, showing information like file sizes and last modification dates, or a cleaned up and filtered list that leaves out dates or some types of files. Along with the directory listing a shopping cart link/button is provided. There may be sub-directories that the user can navigate into and out of. Files that are selectable will have a check box next to each. An "Add to Cart" check box will be provided to add selected files to the shopping cart. Items already in the cart will have a "Remove from Cart" check box.

Cart page

Choosing the shopping cart link, the user will see a cart contents page, showing the selected data files, with perhaps a total download size summary. Check boxes and other controls will be provided so the user can remove some or all of the files from the cart. An "Order" button will be provided to begin the process of gathering the files and packaging them for download.

Order page

After clicking the "Order" button, the user will be presented with an instruction page allowing them to back up to a previous page to continue "shopping" or proceed with the order. This page includes a form to accept an email address to notify the user when their order is complete. Many orders will be large enough that the user will not want to keep their browser window open to the download system until the order processing is ready for pickup. A "continue" with order button is provided to store the email address provided and continue on to an order status page. An order confirmation email will be sent; the user must click the link in this email which contains a unique number in the URL to verify that the email address entered is a working address. The URL given leads to the status page. If the email is not verified in 24 hours, the order is canceled.

We will require orders to have an email address. The system will be more resistant to abuse if an address is verified before any work is done.

Status/Pickup page

The order status page will provide information on the current state of the order [a future version may also provide a listing of the files ordered and a status on each (on disk or on tape)]. Once all files are on disk they will be combined into an archive (tar file). The Stornext contents are already compressed, so a compression step is not needed. The status page then serves as a pickup page, with a link to the deliverable tar file. A notice that the order is ready for pickup will be sent to the email provided when the deliverable is ready. If this page is visited after the deliverable has been deleted, this page should note that this has happened. A link can be provided to re-create this deliverable without the user having to remember and rebuild the order from scratch.

In rare cases, a shortage of resources, mainly working disk space, may prevent this order from proceeding, in which case this status page will note that the order is waiting for resources. Additional status about the system itself may be nice, noting if the Archive Retrieval Server retrieval is paused for error/maintenance, and how many jobs are queued ahead of the current order.

Some additional controls are provided:

- A link/button to cancel this order (immediately frees up disk space used so far)
- Once the order is complete, and the user has successfully downloaded the deliverable, there is a "Finished" link the user can click to indicate they are finished (allowing the system to immediately delete the deliverable and free up resources)
- links to other orders this user has placed. If this order is waiting for disk space, then this provides a convenient way for the user to access the status pages of other orders which they no longer need--on those pages they can then use the "Finished" link to free up resources, and possibly bring their current order closer to completion.

Order complete email

The order complete notification email contains a link back to the order status/pickup page described above. The notice contains the size of the deliverable, and notes a time limit the user has to pick up their order before it is automatically deleted to free disk space for other users (it "ages out"). Besides the link to the status/pickup page, this notice should include:

- A link to begin another order

- A link to the domain specific page
- Additional instructions or metadata relevant to the data downloaded
- A link to mark their order as "Finished", releasing disk space before it "ages out" on its own. This is the same as the "Finished" button on the status/pickup page.

2. Internal User Shopping Cart

An internal user, like a data manager, could also prepare data collections using the shopping cart interface. This would also be a web based interaction, using the same software described for use case 1. However, there will be some additional features available to an internal user:

- A login to identify them as a privileged user or a different application URL only accessible to NGDC.
- An option to choose delivery to one of their NFS mounted file partitions (usually NetApp), these should be set up in advance so that the location can be selected from a menu
- A separate workspace to accommodate larger orders than available to the external web users.
- An option to skip archiving the order into a single tar file
- An option to only move data from tape to Stornext disk buffer
- An option to set the order priority as higher or lower than the typical order
- Access to sensitive data (nonpublic)

3. Monitoring

There are two types of administrators, technical system administrators that will want to know performance and load measurements helpful in maintaining the Stornext access system, and NGDC supervisory administrators (senior staff, data managers) that would like to know how much data are being delivered out of the Stornext system. Note that this system does not handle or record data being written into the Stornext storage area, only information read out.

3.1 Technical Monitoring

Those users who maintain the Stornext access system will want to know

- Total data volume moved
- Metrics on time needed, to judge system load
  - binning by hour of day could be helpful
- percentage of hits on disk cache vs. tape access
- internal vs. external (web) originated requests
- disk space utilization
- distribution of job priority
- archive database table sizes
- job database table sizes
  - size of contacts table
- jobs in queue (waiting)
- jobs processing
- current job queue
- number of jobs delivered (since timestamp, last hour|day|week|month, by hour of day|day of week)
- download objects on disk (#/% waiting for delivery and total)
  - average order size
- cancelled order count
- system state (run or pause)
- error report, type, number of occurrences, last date of each type
- NFS delivery partition list and sizes

An administration mode (login required) allows orders to be held, deleted, have priority or delivery destination changed. An administrator may also change the state between run and pause, and re-order the queue.

3.2 Supervisory Monitoring

It is expected that Data managers and Senior Staff at NGDC will also be interested in some of the information above, probably in a more executive summary format. Note that information about the categories of data (MGG, STP, Hazards, etc.) and the domains of external users initiating requests is **not** tracked by this system since that should be adequately handled by the Web server logs and the log analysis software already in place at NGDC.

4. Internal Program API

Software applications (programs internal to NGDC) that need to access the Stornext will also use the API provided to the shopping cart. This will provide similar functionality but in a format more appropriate to automated interaction. If there is additional functionality, use cases for that can be added here.

Specific User Input

I think it will be useful to group automated access notes by data manager. Other data managers can be added as well, I know not all of them are on this list, but these are people that have been recommended as likely to have input on program API functionality. Notes and edits can also be made to other sections, so don't feel restricted to this portion of the page.

- Eric Kihn (DMSP)
  - The basic functionality provided by the system FSL is using would likely meet Eric's needs.
  - Will likely use his current shopping cart application modified to use the new API
  - Would like to have streaming/caching files. We can implement a function to preload the Stornext buffer disk by requesting files that will be used but the user isn't ready for.
- Dan Wilkinson (GOES/POES)
  - *will meet to discuss the retrieval system after AGU*
- Dave Fischman (Multibeam/NOS)
  - Will it be able to pull out files within a tarball or the entire tarball be downloaded?
    - A: No, we don't plan to do this, we will be delivering what is archived in the Stornext. Our API will allow users to perform these types of actions on their own. So it is a good idea to load the Stornext with tarballs of the size you think are useful for retrieval.
  - Will we be able to pass it requests from our existing websites?
    - A: Requests from the existing websites will need to be modified to include an email address for later notification in the case that files are not immediately available on disk. Some of the order processing software that already does this won't look different to the user, but the software will need to be modified to use the new API.
- Fran Coloma (GPS CORS)
  - A flag to mark sensitive data that should not be displayed to the public is important.

    Thanks Ken, I'm looking forward to seeing the API. Some questions...

  1. Will there be metrics to watch for "popular" datasets, and will there be a process to keep these datasets in a "data pool" so that the tape drive isn't hit repeatedly with similar requests?
     - A: The web logs should provide information on what is popular for data managers. On the delivery side, the disk buffer for the Stornext is expected to keep items that are popular on hand for quick delivery. The most recently used items are kept on disk, and the Stornext's internal disk is relatively large, so popular files will always be present unless an extremely large request has been pulled out more recently.
  2. Is the process smart enough to recognize that a requested file may be in the Disk Buffer or the Retrieval Server instead of hitting the tape drive again?
     - A: Yes, the jobs database will know which files are on the Retrieval Server disk. The Stornext automatically looks for files on it's disk before going to the tape library.
- Rob Redmon (Ionosonde)
  - The current holdings of ionosonde data are around 2 TB, in 30-50,000 station days of data, archived and compressed by day. In the future the amount of data is expected to grow exponentially, perhaps 3 times more frequent and 100 times in size.

    - Programming interfaces are of the most interest to Rob, having an API that makes data access easy to perform by a script would be nice. (ruby)
- Dan Metzger (Trackline)
    - A shopping cart that presents choices similar to a directory listing would be similar to what is being presented now for MGG data users.
    - There are README files in the system hierarchy that help a user decipher the filename meanings. It would be nice to have the content of these available during navigation of the directories. It may be possible to have an an excerpt of these (up to 2000 chars would capture most in their entirety) in the files database.

Requirements

Either directly specified or derived from the Use Cases, list the functionality the Archive Retrieval Server API software must fulfill.

- Version 2 will provide an example shopping cart application with a file system type browser.
- Basic facilities to accept filenames, email address and ordering parameters including
    - notify user when job is complete. An email notification may not be desired if a program is submitting orders
    - specify desired action for orders deemed too big
    - priority option for internal users
    - URL to start over on the browse process
    - name to describe order (optional)
- Process orders serially
- Use a shared database table to coordinate with archive software
- Manage disk space for delivery of products
- Keep a history of Jobs
- Possibly be compatible with the NEAAT (CLASS **N**OAA **E**nterprise **A**rchive **A**ccess **T**ool)

NEAAT Compatibility

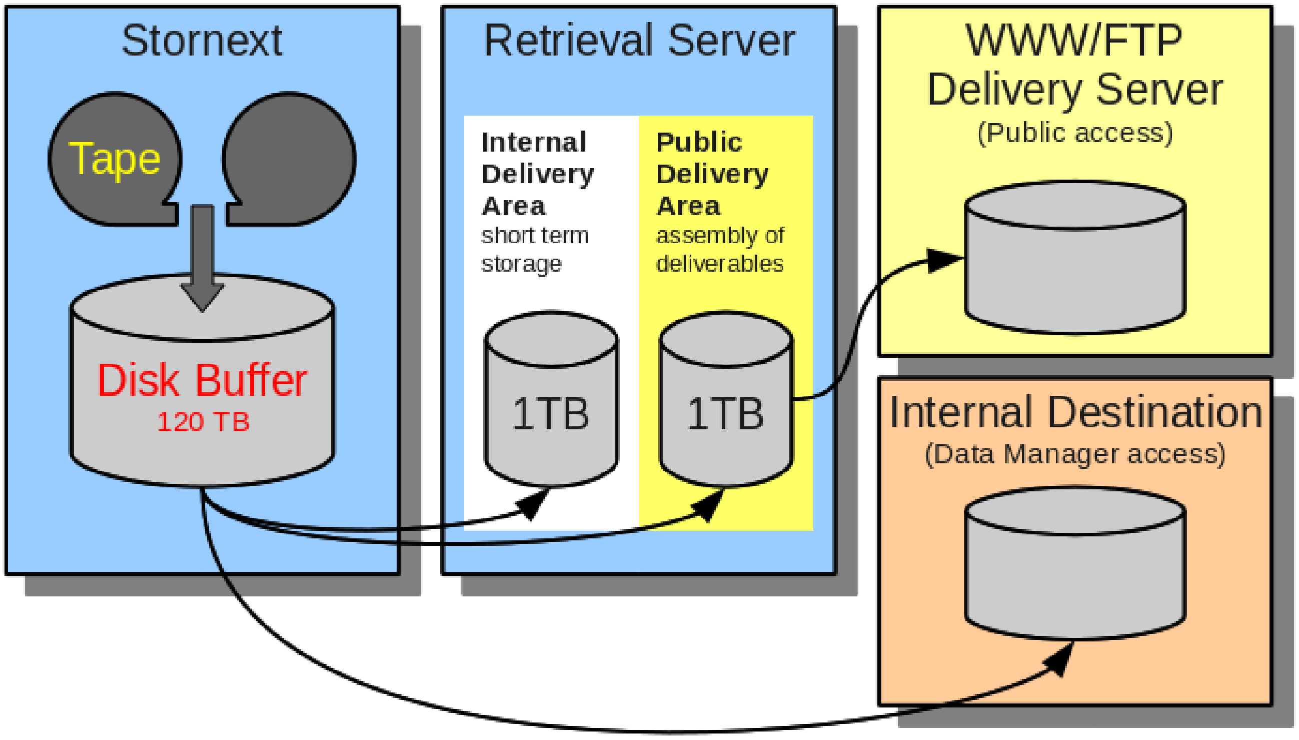
There are some simiarities in purpose between this software and NEAAT. It would be good to provide compatiblity between the Archive Retrieval Server software and NEAAT, to minimize the effort that archive access developers need to expend to switch over from the NGDC archive to NEAAT/CLASS archives.

NEAAT has a higher level of functionality that the Retrieval Server software does not attempt to addresss. Some of the functionality that NEAAT provides that is not in the Retrieval Server plans include:

- Metadata storage and management by data providers
- Service registration and management by service providers
- Data selection by time window and spatial location
- Web Application for data providers to manage their holdings
- Web Portal for Client developers

Design

Hardware Layout



Four main computers are involved. The **Stornext System** runs on its own hardware. A separate **Retrieval Server** will draw data over the network. Once deliverable packages of data are ready, they will be moved onto the public **WWW/FTP Server** for delivery or another **Internal Destination**. Not shown here are databases to track Archive inventory (Archive DB) and orders (ARS DB). Sizes in this section are subject to adjustment based on actual usage needs.

The **Stornext System** has a disk buffer for managing files retrieved from tape. The management of this cache is under control of the Stornext software and is not accessible to the access software in this design.

On the **Retrieval Server**, there are two disk buffers, a 4 TB area for internal users, and a 1 TB work area for collecting files to fulfill a data order for an external customer. (This should probably be a soft boundary of 4 and 1 TB, maybe the server parameters/administration interface can set upper limits.) The collected files will be combined with an archive utility like `tar`, consequently deliverables up to half the size of this area, 500 GB, can be created. For external users, once the order is ready, it needs to be copied off the Retrieval Server Public Work Area out to another system (WWW/FTP Delivery Server), allowing the disk area to be cleared for processing the next order. Internal users can leave their files on the Internal Work Area for short term operations such as un-tar and un-compress, but should free space promptly when done.

Disk space on the **WWW/FTP Delivery Server** will be managed by standard file watch and cleanup software, usually clearing files older than a certain age if not deleted explicitly before then.

It is expected that internal users will often want files delivered directly to an **Internal Destination**, which is configured into the jobs database (see the delivery_partition table). Besides the Internal Work Area, any NFS mounted disk can be configured to receive data. A parameter to preserve some free space on each destination can be configured in case the system owner doesn't want to make the full capacity available to the Retrieval Server. This fourth box in the diagram represents a category of machines rather than a single system, and can be network attached storage or an NFS capable server.

Software Levels

Three levels of software layers are planned for the Archive Retrieval Server API. At the top are interfaces that the users will use directly, such as a web accessible browse and shopping cart system. The middle layer will manage remote and logical file access, providing a published API for other developers to create other top level data access programs. The middle layer will also manage the orders and their status, using a database to store this information. The bottom layer API will only be exposed to the middle layer, and will represent local and physical access type methods.

The interface to the middle layer will be REST, allowing network access through HTTP.

Functionality Outline

Grouped by software levels, here is the functionality to be provided in pseudo-code like entries.

Upper Level Capabilities (Shopping Cart)

The browsing features will be in version 2 of the retrieval software. For version 1 just order support will be implemented.

These are capabilities that end users will see and interact with. Web server applications will provide these services and will use the middle level API to see and retrieve data.

View Directory
: Show a directory of files, like the unix 'ls' command. If a directory location is omitted, then show the root level of the storage system. The user can then navigate into and out of subdirectories as well as select file entries to be added to a shopping cart.

Show Cart
: See the contents of the shopping cart, this is a list of items selected for download. These can be removed if unwanted. Heuristic, put the most recently added item on top, allow viewing with other sort orders as well.

Add File(s) to the Cart
: Add the selected file(s) to the cart

Remove File(s) from the Cart
: Remove the selected file(s) from the cart

Clear Cart
: Remove all the cart contents

Set Contact Info
: Store email address and an optional name for this user. This email address will receive the notification that the data are ready for pickup.

Checkout
: User indicates that data selections are done. Add order to queue for processing

Show Jobs
: Display a status page showing the state of this user's order(s)

Login/logout
: Access control for privileged users

Cancel Order
: Allow user to cancel their order

Immediate file retrieval
: If the file is on disk, return the file, if not, an error is returned (this does **NOT** initiate a retrieval from tape). Can be accessed as a REST URL.

Shopping Cart research

See Shopping Cart Research

Middle Level Functions (Retrieval Server)

These are functions provided to the Upper level to support the listed Upper level capabilities. These functions interact with the Jobs DB to track and provide information on data requests. There are two categories of Middle Level functions, those functions that support external and internal users, and those that serve system administration.

User Functions

These functions support browsing and ordering.

String[] Browse.list(String dirPath, [int jobId])

Get a list of files from a directory, like the unix 'ls' command. If 'dirPath' is omitted, then start at the root level. If a jobId is provided, then items already in the shopping cart can be marked.

int Browse.listCount(String dirPath)

Get a count of the files and directories inside the given 'dirPath' (same as the length of the listing).

String[] getCart(int jobId)

See the contents of the shopping cart, this is a list of items selected for download. These can be removed if unwanted. Heuristic, put the most recently added item on top, allow viewing with other sort orders as well.

int Order.addToCart(int jobId, String[] fileList)

Add the selected file(s) to the cart, if this is a new cart, jobId should be null, and a new jobId will be created.

int Order.addToCart(int jobId, String dirPath, String fileRegEx)

Add files to the cart using a regular expression to match filenames, if this is a new cart, jobId should be null, and a new jobId will be created.

String Order.validateFullOrder(String email, boolean notifyFlag, int acceptanceCode, String name, String orderType, String[] fileList)

Can be called before calling Order.fullOrder with the same arguments to verify the order is valid. Returns a string "ok" for acceptable orders or an error message if an order will be rejected. The error messages indicate why the order would be invalid:

- Error: invalid email address syntax
- Error: unknown acceptance code
- Error: empty file list
- Error: invalid filenames in file list, file names: file1, file2, up to file10 (ends in "..." if list is incomplete)
- Error: filenames not in archive inventory: file1, file2, up to file10 (ends in "..." if list is incomplete)
- Error: order is too large (for orderType)

int Order.fullOrder(String email, boolean notifyFlag, int acceptanceCode, String name, String orderType, String[] fileList)

Add the selected file(s) to the cart and checkout all in one step. All information needed for a complete order will be included in this method call. The arguments are:

email
: Email address for notifications, normally for job status

notifyFlag
: Suppress email to the email address, allows emails to be suppressed for orders initiated by a program

acceptanceCode
: Normally all valid jobs are accepted, but those that are too large are kept in a 'held' status so as not to overload the system. An administrator can review the held jobs. Acceptance codes other than '0' indicate special handling:

    - 0 - Normal processing, this code should be used by default. Valid jobs are set to "paused" if too large.
    - 1 - Reject rather than accept a job that is too large. Can be useful for automated processing to avoid filling the job queue with held jobs.

    - 2 - Accept any valid job, but set its "Paused" flag is true regardless of size.

name
: Name of order to be used as the root of the tarball delivered. This will also appear in the subject line of notifications, allowing users to filter their email on this name if desired. This should be a single word using characters valid for filenames (a-zA-Z0-9_.+-). Invalid characters will be replaced with underscores.

orderType
: Type of order, "internal" or "public", affects permission to access "nonpublic" files

Returns a newly created jobId.

Possible faults:

1 - Invalid email syntax.
: If the email does not follow the format of "name@somewhere".

2 - Files not found.
: Some or all of the files requested were not in the archive Files table.

3 - Restricted Files.
: The user was denied access to some or all of the files requested.

int newCart()

Create a new empty cart, returning a jobId. addToCart() can also create a cart if called with a null jobId, but this function allows this to be done without adding a file to the cart.

removeFromCart(String[] File, int jobId)

Remove the selected file(s) from the cart

clearCart(int jobId)

Remove all the cart contents

setContactInfo(String email, String name, int jobId)

Store email address and an optional name for this user. This email address will receive the notification that the data are ready for pickup.

int sendConfirmationEmail(int jobId, String url)

Send a confirmation email to verify email and confirm job. URL is the web address to receive the job code.

checkout(int jobId)

User indicates that data selections are done. Add order to queue for processing, but set state to "EmailPending"

int confirmEmail(int jobCode)

job code confirms email, gets corresponding jobId, and moves job to "Ordered" state. Returns jobId

String[] getJobs(int userId)

Return a list showing the state of jobs owned by a single user

String[] getJobFileState(int jobId)

Return a detailed list of files in a given job showing status (name, size, onDisk)

String getJobState(jobId)

Returns status on the job, Building, Ordered, Queued, Processing, Retrieval Completed, Delivery Packaged or Order Completed, and any sub status.

int getJobPriority(int jobId)

Get the current priority number

String[] dirList(String dirPath, String regex)

Get a directory listing starting at 'dirPath'. If given, apply a regular expression filter to the file names.

int dirListCount(String dirPath, String regex)

Get a count of the files that would match the corresponding dirList command.

startRetrieval(String destDir, int jobNo, int priority)

Begin processing the cart associated with 'jobNo'. Regular users can set a lower priority than normal. Only privileged users can set a "High" priority job. For external users, create an archive of files, uncompressed, since the objects in the Stornext are already compressed. When done, the result is moved to the destDir and a notification is emailed, if contact information is available.

cancelOrder(int jobId)

Allow user to cancel their order. Set status to complete, leave jobFinishDate as NULL.

getDiskFile(String filename)

Supports the Immediate File Retrieval capability above. Gets a file from disk, if the file is on tape return an error but does NOT initiate a tape access.

preloadFiles(String dirPath, String fileRegEx)

Request that the Stornext move files matching the given file pattern from tape onto buffer disk. This attempts to expedite later retrievals by pre-caching files onto the Stornext disk buffer. You should only request files that you actually plan to use.

Administrative Functions

These functions support maintenance and administration.

TODO: flesh this out from the technical monitoring capabilities needed.

String[] listJobs(bool internalUser, status filter)

Retrieve a list of jobs, sorted by priority and order date. The list may be implemented a s list of job objects instead of strings. The list will show sizes, priorities, status, and dates started, ordered, processing, and complete. Normally jobs that are completed will not be in this list, but any combination of status, Building, EmailPending, Ordered, Processing or Completed can be specified. If this is from an internal user, then originator contact information will be shown.

String[] partitionFreeSpace()

Reports on space available on the predefined delivery partitions. The internal work areas are also included in this report.

bool setJobPriority(int jobId, int newPriority)

Alter the job priority if this is a privileged user. Returns True on success

bool setRunStop(string "run" or "stop")

Set the retrieval server to running or stopped state. Incoming user requests when the retrieval status is "stop" will get a fault result "Server is stopped".

Lower Level Functions (Archive Server)

These functions are available to the middle level API. These functions may interact directly with the Stornext SNAPI software and command line interfaces.

Communications with the Archive Server is through the **`archive_interface`** table in the jobs database. The database to use is:

- **adic_jobs** for development
- **ars** for production

Details of usage can be found in the archive_interface table description

String archiveStatus()

Check on the the Archive Server system. This should indicate if the system is "running" or in "maintenance" mode. File retrieval requests will most likely be refused when not running.

Sample response:

```
<params>
  <param>
     <value><string>running</string></value>
   </param>
 </params>
```

struct fileStatus(String filePath) -- Version 2

Return file information including size, modification time, last access time and onDisk status. This information is also updated in the Archive Database for this file when this method is called. If not too costly, mediaID could also be returned, but this is not needed for the first revision.

Sample fileStatus result:

```
<struct>
  <member>
    <name>size</name>
    <value><i4>filesize</i4></value>
  </member>
  <member>
    <name>modTime</name>
    <value><dateTime.iso8601>YYYYMMDDTHH:MM:SS</dateTime.iso8601></value>
  </member>
  <member>
    <name>lastAccessTime</name>
    <value><dateTime.iso8601>YYYYMMDDTHH:MM:SS</dateTime.iso8601></value>
  </member>
  <member>
    <name>onDisk</name>
    <value><boolean>1 or 0</boolean></value>
  </member>
</struct>
```

Security Issues

If there are some files in the Stornext archive that should not be publicly accessible, then they should be placed in or below a directory named "**nonpublic**" (all lowercase). Because data managers can move files in or out of these directories should their status change, it is believed this is a sufficiently flexible mechanism that is easy to use.

The "nonpublic" directory can be anywhere in the path. For example, if the directory is:

```
/a/b/c/nonpublic/d/e/file
```

Then everything from "/nonpublic/" and below is not shown to the public. Submitting this path to the directory listing function will give a file or directory not found error, with possibly a suggestion that the closest available directory is "/a/b/c"

There is also a "**private**" security level, that internal users can see, but cannot access directly. Data managers must ask an administrator to retrieve these files.

Do we need a "**nobrowse**" security level that hides files but allows them to be put into the cart if asked for by name?

Optimizations

Some possible design features to increase performance.

Segment Big Orders

Big multi-tape orders can be hard to interrupt for system administration, or interleaving jobs. Large orders can be broken into 1-tape-at-a-time chunks, allowing better responsiveness for an administrator that wants to pause the system, or manipulate job positions or priorities. A tunable parameter can set a number of maxFilesThreshold (300 files will not overwhelm the command line used for fsretrieve given the current database directory column size of varchar(256) and file size varchar(100)). Another tunable parameter can set the upper maxRequestSizeThreshold, so 100 GB and larger orders will be segmented into smaller numbers of files to get under, if possible, 100 GB per fsretrieval.

Aggregate Small Orders

Along with segmenting big orders, it may speed things up if small orders collected out of the queue to try and meet some tunable parameters of minFilesGoal (250 files?) and minRequestSizeGoal (10 GB?). I'm calling these "goals" since not making the minimums will not delay the orders, processing will proceed with what is queued.

Lower Priority on Subsequent Orders

For a user that already has entries pending in the queue, subsequent orders are set to a lower priority so that other users can get prompt service. See the priority codes in Archive_Retrieval_Server_Level_1_design#priority_table

Reshuffling requests by Tape

Fetching a tape has the most time overhead. A possible speed up will come from looking through the queued orders, identifying the tapes that contain each file, and then requesting files that are on the same tape at once, even if they come from different orders. Not all the orders need to be scanned, we can look through N files or M MB of data for tape IDs, whichever is less. Setting N and M very large would be similar to scanning through all files requested for tape information.

Let the Stornext optimize retrievals

The Stornext system will gather files off one tape all together and in the right order if they are requested at the same time. It should also be relatively harmless to ask for something that is already on disk, or re-request it. What if we request everything in the queue at once, letting the Stornext pick the most efficient access pattern. Some sort of size limit would be needed.

Database Tables

Two databases are involved in this system. One is an Archive Database, which mainly tracks the state of files so that the Stornext itself does not need to provide this information. The second is a Jobs Database, which will track data orders going to users and provide summary information on Stornext usage.

files table

Files in the archive, stored in the Archive DB.

files (Archive DB)

Column	Type
file_id	int primary key
dir_id	int foreign key
file_name	varchar
compressed_size	bigint
uncompressed_size	bigint
tape_count	tinyint
archive_date	datetime

The filename is a joined to the entry in the Directory table to produce a full path in the Stornext filesystem.

File Metadata columns

compressed_size
    The file size in bytes. Files are archived in a compressed state
uncompressed_size
    The expected uncompressed size of the archive file
archive_date
    Timestamp that the file was added to the archive

dirs table

Directories are stored separate from files in the Directory table for efficiency and good database normalization. Note that this table only contains directory entries which contain files. There are no directory entries that only contain subdirectories. Also, the path entries in this table to **NOT** have a trailing '/' character.

dirs (Archive DB)

Column	Type
dir_id	int primary key
dir_name	varchar

The **dir_name** field is not constrained to be unique, although it may be useful to have it unique.

active_jobs table

JobRecord table

Stored as **ARS_JOB_RECORD** in the database.

Information to track **current** orders of files, contact information, priorities, and status. This is stored in the Jobs DB.

JobRecord (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
delivery_name	varchar (user chosen deliverable root name)
order_id	varchar unique key (YYMMDD order requested + 36 char random alpha-numeric)
status_id	int unsigned foreign key
paused	int (boolean 1=true, 0=false)
debug_me	int (boolean 1=true, 0=false)
err_state_id	ErrState foreign key
priority_id	int unsigned foreign key
[*orig_priority_id*]	int unsigned foreign key
contact_id	int unsigned foreign key
notify_flag	int (boolean 1=true, 0=false)
order_type	varchar ("internal" or "public")
size	float (compressed size in MBytes)
uncomp_size	float (uncompressed size in MBytes)
file_count	int (number of files requested in this order)
dest_dir	varchar (Delivery Server directory, see delivery_partition table)
deliverable	varchar ("tar", "dir", or "cache")
job_start_date	date (item added to cart, first date)
order_date	date (datetime order requested, second date)
queued_date	date (datetime email verified by user, third date)
proc_start_date	date (datetime processing started, fourth date)
archive_finish_date	date (datetime tape archive retrievals finish, fifth date)
proc_finish_date	date (processing done, data moved to delivery and user notified, sixth date)
pickup_date	date (User used pickup URL, seventh date)
delivery_expiration	date (date that this Job is considered completed or canceled, eighth date)
[compression]	varchar (optional compression type "gzip", "bz2"...)

[description]	varchar (optional user notes to describe order)
restart_url	varchar (optional link back to cart interface)
GORM	hasMany(dataFiles:DataFile, retrievals:Retrieval)

Columns in [square brackets] can probably be left out of the first version of this software.

priority_id
: is the effective priority of the job. This will differ from origPriorityId if an administrator changed the priority or if the Lower Priority on Subequent Orders optimization has modified the priority.

orig_priority_id
: is the original priority of the job when created.

notify_flag
: is used to turn off email notifications, typically when the requestor is a program.

deliverable
: If this is "**tar**", which is always used for "public" order_types, then files are tarred into a directory named after the order_id. This can be "**dir**" for an internal user, in which case the 'dest_dir' and 'delivery_name' are used to determine destination. If this is "**cache**", also for internal users, then files are left on the Stornext disk buffer, it is assumed that the requester is internal and has a way to access files directly from this location.

dest_dir
: is an NFS mounted filesystem (NetApp) location in the delivery_partition table. This value is copied from the delivery_partition.path column and is not a foreign key id. The reason this is not a foreign key is to allow us to change values in the delivery_partition table without altering the job history on partitions used.

delivery_name
: is used in the subject line of notificaiton emails, allowing the user to filter emails if they desire. For internal orders, this will be used as a directory name to receive files and should be a valid file path specification

Some possible statistics that can be calculated from this table:

- volumeTotal (in GB)
- volToday (in GB)
- volThisWeek
- volThisMonth
- cartsStarted
- ordersPlaced
- filesRequested
- filesOnDisk (when requested)
- priority3Count
- priority2Count
- priority1Count
- internalOrders
- ordersCanceled
- ordersCompleted
- maxJobQueueTotal
- maxJobQueueWeek
- maxJobQueueMonth
- timeToOrder (seconds from start of building to Ordered on average)
- timeInQueue (seconds)
- timeInQueueDiskWait (seconds waiting for disk)

status table

Job status states used by the **status_id** field in the active_jobs table and the job_history table

status (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
name	varchar unique

Job Status States

These are sequential states, so normally a job in any state has also passed through all the previous lower numbered states in order. An exception is when jobs are canceled and may thus skip to the **Completed** state. If substates are implemented, then these skips can be detected.

1 Building
The user is building their order. A non-empty list of files (shopping cart) has been started, this is the first state a job enters when it is created. Synonyms: Shopping, Pending, session Started.

2 Ordered (Email verification is pending)
An email was sent to the address entered, but the user has not yet clicked the enclosed link to verify that the email is valid. This verifies the email works and has the email recipient confirm that the order should begin proccessing. When Ordered, we verify inventory by provider, check permissions, and build retrieval records.

3 Queued (Email has been Verified, Waiting to be processed)
Data selections are complete, the user has requested this set of files and the job is waiting for processing.
Substates:

- Waiting for space on *partition*, *n* MB available - order is held up waiting for space on a destination disk.
- Waiting for manual intervention - order was too big, user has asked for a data manager to complete the order.
- Held by administrator - an adminstrator has put this job on hold.

4 Processing (Rename 'Retrieving'?)
Retrieving data files from tape, the Order Manager processing this phase.
Substates:

- Tape *n* of *N* - (for large orders) break big orders up and submit file requests by tape ID.

5 Retrieval Completed
Tape retrieval is complete. Packaging, clean-up, moving to delivery. The Delivery Processor is working on the data.
Substates:

- Preparing files - tar or copy in progress
- Data transferred - delivery package moved to pick-up area

? Should there be a 'Packing' status?

6 Delivery Packaged
A delivery object is ready for pickup at the pick-up point (delivery directory) specified.
Substates:

- Awaiting pickup - the user has been notified
- Download attempted - the download URL was used (don't know if it was successful though)

7 Order Completed
Orders in this state should be in the Job History table, and may be purgeable.
Substates:

- User released storage - the user indicated they were done with the files.
- Delivery window closed - the delivery files were automatically deleted after the end of the pick-up time window was passed.

- User canceled order - the user canceled the order while waiting to be processed.
- User canceled processing - the user canceled the order after processing began.
- Administrator canceled order - an administrator canceled the order while waiting to be processed.
- Administrator canceled processing - an administrator canceled the order after processing began.

priority table

Job Priorities. We will start off with a simple priority system, with jobs being created with medium priority by default. High priority jobs are completed before Medium jobs if resources are available. Low priority jobs run when there are no other jobs waiting. Jobs at the same priority level are taken in the order queued.

If the Lower Priority on Subsequent Orders optimization is in effect, then the first of the low priority jobs for a given user will be elevated to medium (going to the end of the medium priority queue) every time a new job is needed. Priorities 1 and 5 are not optimized, so the administrator has the final say by using one of these.

priority (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
name	varchar unique

Priority Numbering:

1. Highest (Administrator only can set this)
2. High (internal user option)
3. Medium (internal and external users)
4. Low (external users subsequent orders)
5. Lowest (Administrator set)

contact table

For jobs that will take some time, the initiating user will submit an email address and then receive a notice when the order is ready for pick-up.

contact (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
email	varchar
last_used	date (for clearing out old external contacts)
[name]	varchar
[internal]	boolean (true if NGDC internal user)
[email_verified]	boolean (true if confirmation email not needed. for internal users only)
[password]	varchar (for internal users only)

next_jobs table

A bookkeepping table to maintain the order of the next job to process. To be replaced by a RetrievalQEntry and DeliveryQEntry table, both based on a QueuedThing.groovy base class.

next_jobs (Jobs DB)

Column	**Type**
id	int unsigned auto_increment primary key
job_id	int unsigned unique foreign key

provider table

A table listing the different data providers, either NGDC or NEAAT.

provider (Jobs DB)

Column	**Type**
id	int unsigned auto_increment primary key
name	varchar

retrieval table

A retrieval_id generation table to associate retrievals to a job_record.

retrieval (Jobs DB)

Column	**Type**
id	int unsigned auto_increment primary key
procStartDate	timestamp
procFinishDate	timestamp
provider	Provider foreign key
thread	int

GORM	hasMany(dataFiles:DataFile)
	belongsTo(job:JobRecord)

Usually there is one retrieval ID per JobRecord, but since a job could have more than one retrieval (e.g. batches for a large number of files), there may be multiple entries of the same job_record_id value in this table, each with its own unique (retrieval) ID.

This class keeps track of active retrievals. A retrieval can be in one of three states:

- queued - waiting to be retrieved (procStartDate == null, thread == null)
- locked - actively being retrieved (procStartDate != null, thread != null, procFinishdate == null
- done - processing completed (procFinishDate != null)

retrieval_and_file table

A junction table to tie files to a FS retrieval. This table and the archive_interface table are used to communicate retrieval requests to the Archive Server. This table contains retrieval file data, the archive process only read this table.

retrieval_and_file (Jobs DB)

Column	**Type**
id	int unsigned auto_increment primary key

retrieval_id	int unsigned foreign key (FS_RETRIEVE request)
file_id	int unsigned foreign key

The **retrieval_id** can group files if more than one request should be made for a job, for example, 10,000 files would be 10,000 lines in this table, all with the same jobId, but 5 retrievalIds of 2,000 files each. In the future, we may know the media id, and can segment the retrieval based on which tapes the files are stored. Since there may be a need for more than one retrieval_id per source_request entry, the source_request_id is not used to group the files of a retrieval. Also, to avoid duplications of retrieval_id, the database should generate the retrieval_id as a sequence of serial numbers. See the retrieval table to match retrieval IDs to source_request IDs.

delivery_partition table

Stores known delivery areas, these are set up in advance so that the delivery server has write permission to these areas. These may be NFS mounts to the Retrieval Server.

delivery_partition (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
path	varchar
work_path	varchar
save_space	int (MBytes)
usage_code	enum (http://dev.mysql.com/doc/refman/5.1/en/enum.html) ('pub_web', 'in_tmp', 'in_part')
effective_date	date (this partition became active, ignore if date is in the future)
expiration_date	date (this partition became inactive. This is a current or "latest" entry if this field is null)
default_part	int (boolean 1=true, 0=false)
chmod_options	varchar (used to set access permissions after moving results from work_path to path)
order_file_limit	int (0=no limit)
order_size_limit	bigint (MBytes)
description	varchar
access_token	varchar
[order_id_directory]	int (boolean 1=true, 0=false, Enclose deliveries in OrderID directories. This property should always be true for partitions that have defaultPart=true)
[replaces_id]	int (if this is a replacement, indicates entry this is a replaces. Can be used with effective_date to implement automatic replacements)

The **path** and **work_path** entries are absolute paths to directories. They should start with a "/" and should not end with a "/".

The **save_space** parameter preserves some disk space in MegaBytes, keeping the Retrieval Server from using all of the capacity at this destination.

```
availableSpace = actualDiskFreeSpace - save_space
```

For example, if the destination is /nfs/data/mgg/, the saveSpace parameter is 100, and the 'df' (disk free space system command) shows 500 MB available, then the Retrieval Server will act as if 400 MB are available for use. Orders of 400 MB and larger destined for this area will be held as "Ordered: Waiting for delivery disk /nfs/data/mgg"

The **usage_code** meanings are:

pub_web
: Public Web/FTP access

in_tmp
: Internal Temporary space

in_part
: Internal Partiton

default_part marks this record as a default partition, there should be only one default partition per usage_code.

clean_up_policy table

If the Archive Retrieval Server ends up cleaning up files in the delivery area, then here are some clean-up policy names and meanings. This table is not needed if the standard file aging and deleting already in place is acceptable.

clean_up_policy (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
name	varchar

clean_up_policy Values

1 Keep
: Don't delete

2 Keep until
: Delete after a separately given date is reached

3 DeleteAfterAccess
: Delete after a separately given delay is reached after the last access of the deliverable. The delay gives time for the download to complete.

4 Delete
: Delete the deliverable immediately

system_params table

The Archive Retrieval Server is configured and controlled by a system parameters table. This would be a table of name-value pairs.

system_params (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
name	varchar
value	varchar

Some possible entries (values shown are current system defaults):

- sysState=running (or paused)
- publicSizeLimitMb=2000 (orders larger than this many MB are sent to a data manager)
- publicFileLimit=4000 (orders larger than this many files are sent to a data manager)
- internalSizeLimitMb=2000000 (orders larger than this are set to paused)
- internalFileLimit=8000 (orders larger than this many files are set to paused)
- batchFileLimit=1000 (Limit the number of files we scan for tapeIDs when optimizing tape access on a singe large order, see

Reshuffling requests by Tape)
- batchMbLimit=2048 (Limit the MByte size when scanning for tapeIDs for single large order submission, see Reshuffling requests by Tape)
- statusEmail=Ken.Tanaka@noaa.gov (administrator email address)
- notifyFromEmail=Ken.Tanaka@noaa.gov (notifications come from this email address)
- notificationLevel=debug (choose one of: debug, info, warn, error or none)
- omIdleSleep=60 (Number of seconds the Order Manager sleeps between cycles when idle)
- omShortDelay=10 (Number of seconds in a short delay used by the Order Manager)
- dpIdleSleep=60 (Number of seconds the Delivery Processor sleeps between cycles when idle)
- statusUrl=http://intranet.ngdc.noaa.gov/arsStatus/jobStatus/show (URL for the status/pickup page, embedded in notification emails)
- pickupWindowDays=4 (Number of days the user has to attempt a download on the completed order)
- doarchivePrefixCutwnloadWindowDays=2 (Number of additional days the user has to complete the download when they pick up the order)
- diskReserveDefault=200 (Default disk space to reserve if not specified in the delivery partion table)
- jobFileHistoryDays=365 (Number of days to keep file data for an order. Once purged, the order cannot be resurrected by changing its status back to "Queued")
- archivePrefixCut=/stornext/ngdc/archive/ (Common portion of filepath prefix to remove from retrievals (should end in a /))
- deliveryPrefixCut=/stornext (Common portion of filepath prefix to remove from delivery hierarchy (should not include a trailing /))

Maybe add?

- omPriorityPolicy or dpPriorityPolicy as one of
  - Interleave
  - InterleaveMedium
  - FIFO (straight order received within priority)
- extraDownloadTimePerGB=5 hours, so a 10 GB download gets an extra 50 hours added to their download window.
- dp/omCycleSkip=0 # of frequency cycles to skip before really doing anything, used to throttle back quartz processing activity.

retrieval_stats table

Retrieval statistics table. This would be a table of name-value pairs.

retrieval_stats (Jobs DB)

Column	Type
id	int unsigned auto_increment primary key
name	varchar
value	varchar
last_reset	timestamp

Some possible statistics:

- partitionXWaitCount
- systemRunCycles (count of stop to run transitions)
- errorCount
- specialOrderSizeCount (number of orders needing data manager intervention due to exceeding size limit)

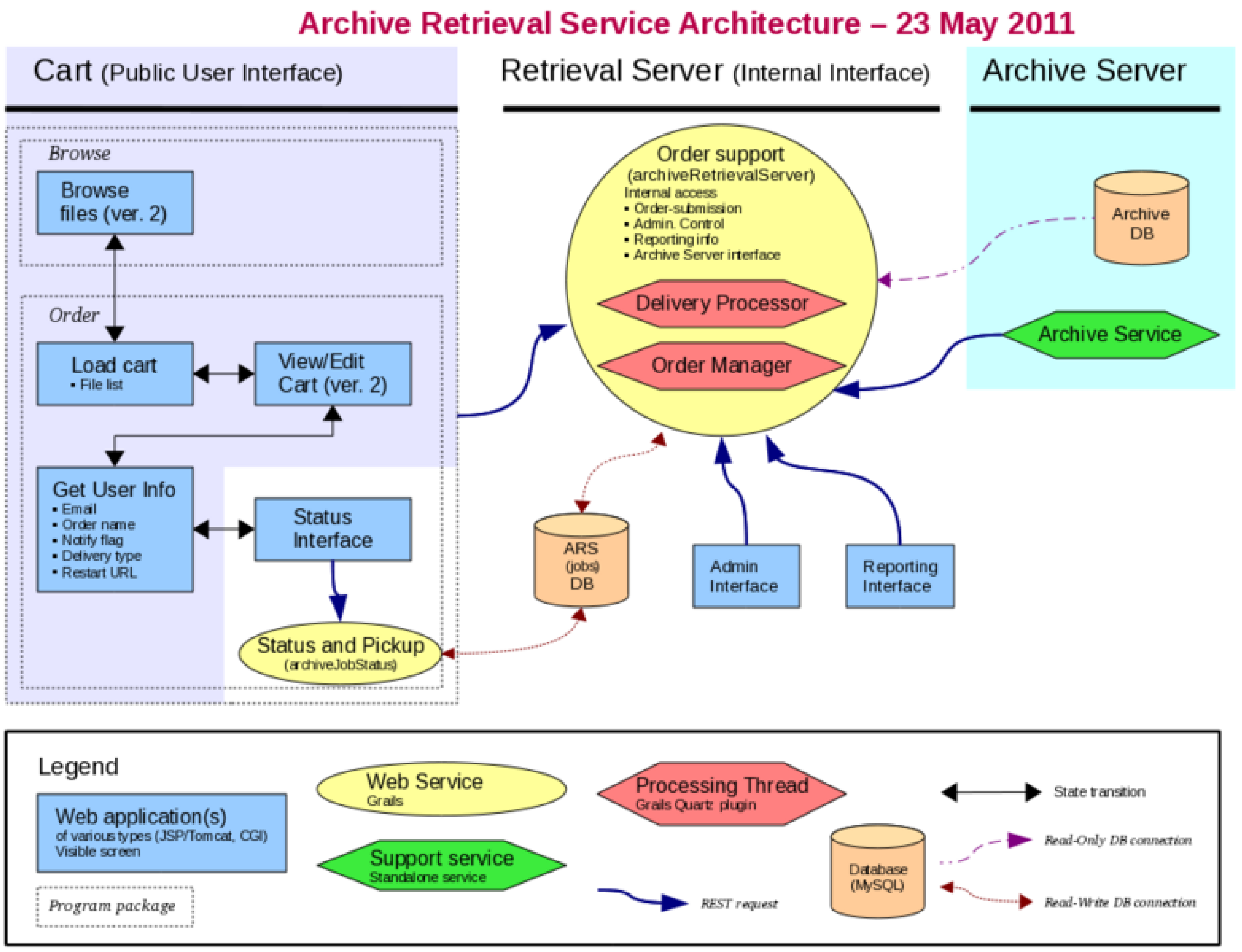
Flowcharts

Some flowcharts are available in the EDS project of SubVersion under https://svn.ngdc.noaa.gov/viewvc/eds/adic/retrievalserver/trunk/src/site/resources/images/

These figures were created as OpenOffice Draw documents and exported to PNG versions for this wiki

Archive Retrieval Service Architecture

This is an overview of the web service architecture, and the expected protocol connections.



Software Components

This is a compact outline of software package responsibilities (functionality) and inter-dependencies. It's intended to be a quick reference to aid in understanding what the components are doing and how they interact. I also note filesystem and database dependencies. ("CRUD", when used below, is an acronym to Create, Retrieve, Update and Delete database records.)

Retrieval Server (archiveRetrievalServer)

- Internal Web Service
  - implemented in Grails
- Database connection by GORM
  - Uses adic_jobs database
  - Uses archive2 database

- Functionality provided by REST
    - Add a full order to active_jobs table in one step
    - Create a new job
    - Add files to job
    - Validation and checks
        - Validates filename syntax (not needed if DB name entries are good)
        - Checks filenames against Archive DB inventory
        - Checks filepaths for "nonpublic" and "private" access (hold violating jobs)
        - Pauses or rejects (based on acceptanceCode) jobs that are too big (Client software can then direct user to a data manager for manual intervention)
    - Creates retrieval lists for Archive Service
        - Organizes jobs by priority, date and number of active jobs submitted by the same user (email address)
        - Interacts with Archive Service to retrieve files from tape onto disk buffer
        - Tracks and updates job retrieval status
    - Marks completed retrievals by setting job state to ' Retrieval Completed'
    - Retrieve job information by ID
    - Retrieve job information by Order ID
    - Retrieve job files by ID
    - Retrieve job files by Order ID
    - Set the substatus of a job (by ID)
    - Set the job to "Download attempted" (by ID) also extends deliveryExpiration by downloadWindowDays time
    - Provide file and subdirectory listings for a directory name
- Interactions with other components
    - arsStatus needs this to be running to provide job information and record download attempt
    - adicOrderManager needs this to provide jobs to process in ' Queued' state

Order Manager (archiveOrderManager)

- Sub-process of archiveRetrievalServer
    - implemented as a Grails Quartz thread
    - handles the quicker schedule-based maintenance tasks
- Database connection by GORM
    - Uses adic_jobs database
    - Uses archive2 database
- Functionality
    - Frees resources for canceled jobs
    - Calculates retrieval statistics
    - Emails administrator any warning conditions or jobs paused due to size limit violations
    - Performs automated DB table maintenance
    - Performs automated filesystem maintenance
- Interactions with other components
    - Depends on archiveRetrievalServer for jobs to process, works on jobs in ' Queued' state
    - DeliveryProcessor needs this to provide jobs to process in ' Retrieval Completed' state
    - Depends on ars to set system sysState to one of 'running' or 'paused'

Delivery Processor (archiveDeliveryProcessor)

- Sub-process of archiveRetrievalServer
    - implemented as a Grails Quartz thread
    - handles the longer running tar tasks
- Database connection by GORM
    - Uses adic_jobs database
- Needs access to shared files systems (NFS, netApp) and Stornext filesystem

- Functionality
    - Verifies disk space available before processing jobs
    - Files for external customers are tarred to the FTP location
    - Files for internal customers are moved to their chosen NFS location
    - User is emailed (unless notifyFlag is false) that their job is ready. For external users, a link to the Job Status page is included.
    - Marks completed packaging by setting job state to ' Delivery Packaged' and substatus of 'Awaiting pickup'
- Interactions with other components
    - Depends on OrderManager for jobs to process, works on jobs in ' Retrieval Completed' state
    - Depends on adicRsAdmin to set system sysState to one of 'running' or 'paused'

Administrator (archiveRsAdmin)

- Internal Web Service, part of the archiveRetrievalServer
    - implemented with Grails
- Database connection by Grails GORM
    - Uses adic_jobs database
    - Uses archive2 database
- Functionality
    - Allows administrator to inspect and change system operating paramters
    - Shows system health
    - Allows admin to view and edit job priority, and other job info
    - Allows admin to cancel a job
    - Allows admin to CRUD delivery partitions
    - Allows admin to CRUD other admins accounts
    - Allows admin to pause or shutdown retrieval activity (by setting adic_jobs.system_params table value for sysState)
    - Performs manual DB table maintenance
    - Performs manual filesystem maintenance
- Interactions with other components
    - OrderManager watches for sysState changes, which can be set by this program
    - DeliveryProcessor watches for sysState changes, which can be set by this program

Job Status/Pickup (arsStatus)

- External Web Service
    - implemented with Grails
- Functionality
    - Allows users to view their job status
    - Allows users to download their job when ready
    - Records user download attempt
    - Allows user to cancel/delete job prematurely
    - Allows user to see their other jobs

Order Manager Description

The Order Manager is a constantly running process. It reads the ActiveJob table and interacts with the Archive Service to complete the current job as far as getting files from tape to disk. Once all files for a request are on disk, the job status is updated and the Delivery Processor is expected to take over on packaging and notifying the requestor.

If there is no current job, then the active_jobs table is consulted to select the next job based on priority and date. After selecting a job, then priorities for the next job may be updated. Loads the retrieval_and_file table and passes a RetrievalId number to the Archive Service. This process will also re-request interrupted jobs.

This process also performs table housekeeping. Completed jobs have their status updated and then are moved from the active_jobs table to the job_history table. This process will also clean old rows out of the retrieval_and_file table.

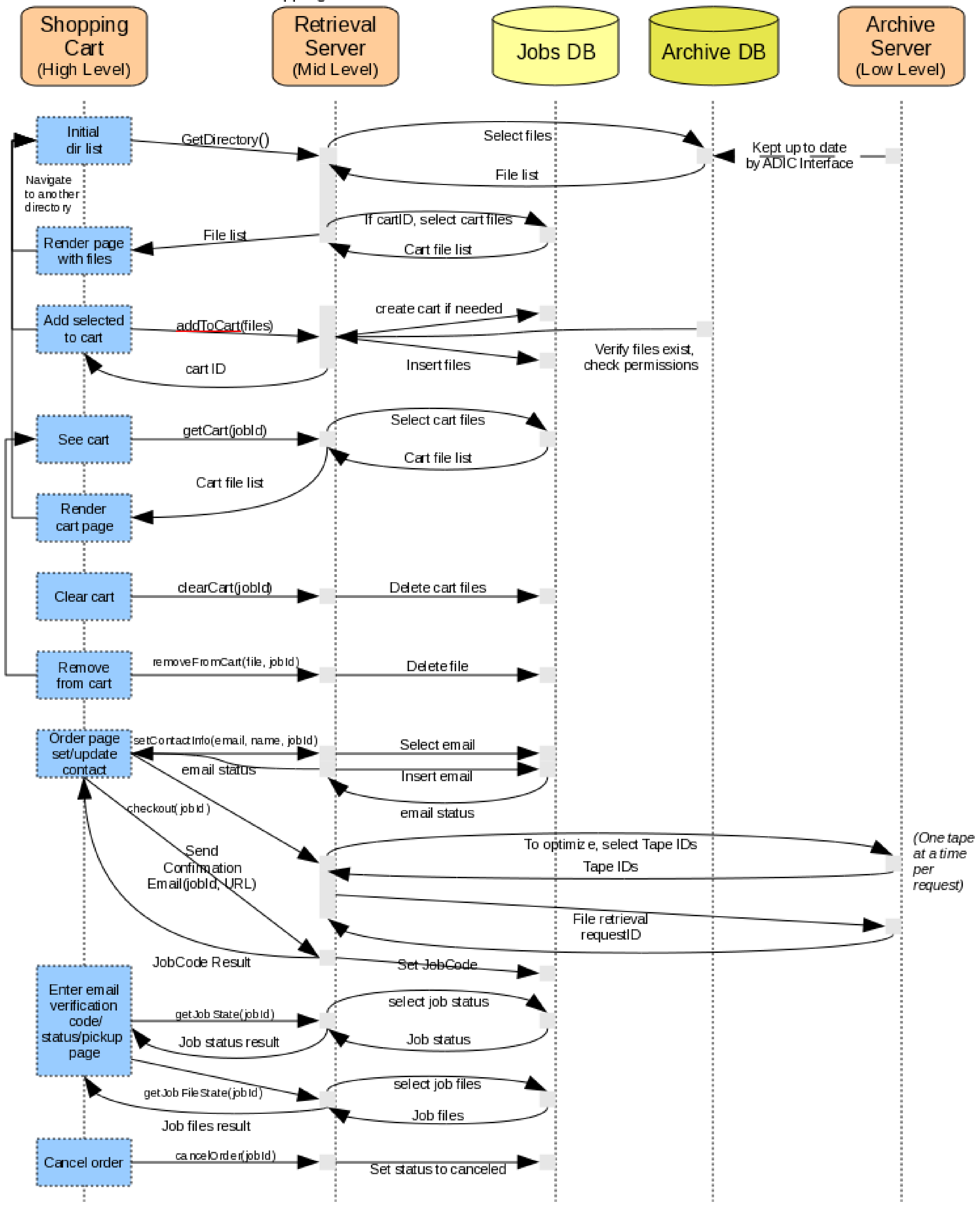
Delivery Processor Description

The Delivery Processor is a constantly running process that completes the current job once all files are on disk. Files are packaged if needed and moved to the delivery destination. The requester is notified and the Archive Service is signaled when the files are no longer needed by the retrieval service.

Shopping Cart Function

Here is an initial draft of the Shopping Cart function flowchart, showing different API level interactions:

Flowchart of Standard shopping cart interactions:



Next Actions

Ken

- Continue to flesh out Mid and Low API calls
- Write Java-to-Java XML RPC test
  - Verify data scaling, 100, 1000, 10000 files returned
    - In Java server to Java client, this worked well on the same system (localhost) using a Tomcat Servlet, 1,000,000 filenames took 3s to transfer. 3M filenames took 11s. Described the code in the Apache wiki: http://wiki.apache.org/ws/XmlRpcExampleStringArray
    - if there's time try this in Ruby-to-Ruby and Java-to-Ruby.
  - Start with a Google on "xml rpc" (http://www.google.com/search?q=xml++rpc&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a)
  - or Google on "xml-rpc java example tutorial" (http://www.google.com/search?q=xml-rpc+java+tutorial&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a)
- Build DB schema
- have the software completed for the all of the Cart Function Calls by end of August. We'll have a check-in mtg mid August.
- Set up **`adic_jobs`** database to store job information as part of the function calls.
- Begin designing a monitoring interface
- Milestones for coding
  - Architecture Overview - Done
  - Implement Database Schema - Done
  - Evaluate iBATIS 2 for database access - Done, iBATIS will be acceptable
  - Primary software packages
    - Administration (Maintenance) Interface - In progress
    - Order Support Interface - In progress
    - Cart order Interface (test page) - In progress
      - Load Cart (submit full order) - done
      - Status/Pickup page - done
    - Order Manager service - done
    - Delivery Processor service - done
    - Browse Support Interface (ver 2)
- NGDC look and feel plugin for Grails code (Jordan has a plugin) - Done
- Security plugin using Crowd for Grails - Done
  - See CrowdedSpringSecurity and Atlassian Crowd Tips
- DB connection pooling
  - See google "Grails connection pool"
    - http://sacharya.com/grails-dbcp-stale-connections/
  - See John C's links
    - DbcpInfo
    - FailoverJDBCConnectionPool
- Redesign Database schema
  - Talk to John L for design suggestions - Done
  - Use NamedThing pattern - Done
- Consolidate code into Grails, using the Quartz plugin (http://grails.org/plugin/quartz) for scheduling

Tom

- Check into SNAPI v 2.0.1 for StoreNext 3.1.2
  - Can it handle multiple files? API implies single file for FileRetrieve, command line fsretrieve says "Separate multiple file names with a space."
- Document Archive tables "File" and "Directory"

Dan

- Ask data managers about workability of 'non_public' directory to hide data.

See Also

Web Access to ADIC Planning Page

Archive Retrieval Server Development Notes

- REST interface to the Archive Server
- Archive Retrieval Server Test Plan

Previous, now obsolete, ADIC Level 0 design intranet wiki page

Archive System at NGDC, see the ADIC section

ADIC Stornext documents online (http://stornextnotes.blogspot.com/2008/06/pdfs.html)

ADIC-API Project Gantt chart (http://intranet.ngdc.noaa.gov/Archive/adic-api/adic-api-chart.html) and ADIC-API Project Task list (http://intranet.ngdc.noaa.gov/Archive/adic-api/adic-api-tasks.html)